CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 1 – Nov. 2019

Sub:	Unix and Shell Programming							Sub Code:	18MCA12
Date:	16/11//2019	Duration:	90 min's	Max Marks:	50	Sem:	I	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE	
			CO	RBT
1	Explain the below mentioned commands with its usage and examples. i) script ii) uname iii) spell iv) bc v) who	[10]	CO5	L2
	OR			
2(a)	State and explain the types of shell variables in UNIX.	[7]	CO5	L2
(b)	Write a note on here document.	[3]	CO5	L1
	PART II			
3	With a neat diagram, explain the architecture of Unix operating system.	[10]	CO1	L2
	OR			
4a)	Briefly explain for loop in shell script with examples.	[5]	CO5	L2
b)	Explain different forms of if statement with an example.	[5]	CO5	L2
	PART III			
5a)	Differentiate between hard link and soft link	[5]	CO5	L2
b)	With suitable example bring out the difference between absolute and relative path names.	[5]	CO5	L2
	OR			
6.a)	Briefly explain the unix file system.	[6]	CO1	L2
b)	What is a file? Explain the categories of files found in UNIX Operating System.	[4]	CO1	L1
	PART IV			
7	Write a shell script to find the given input character is an uppercase, small case, digit or special symbol using case conditional statement.	[10]	CO5	L3
8.a)	What are attributes? How to list file attributes and directory attributes? Discuss with examples.	[5]	CO1	L2
b)	Explain how one can change the permission of a file in detail.	[5]	CO3	L2
	PART V			

9	Discuss the types of tests performed by the test command.	[10]	CO5	L1
10	Describe the important features of the UNIX operating system	[10]	CO1	L2

CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 1– Nov. 2019

Sub:	Unix and Shell Programming					Sub Code:	17MCA12	Branch:	MCA
Date:	16/09/2019	Duration:	90 min's	Max Marks:	50	Sem	V		OBE

1. Explain the below mentioned commands with its usage and examples.
i) script ii) uname iii) spell iv) bc v) who

script:

The ***script*** command is used to record the session in a file. When you have are doing some important work, and would like to keep a log of all your activities, you should use ***script*** command immediately after logging in. For example,

$script
Script started, file is typescript
$

Now onwards, whatever you type, that will be stored in the file *typescript*. Once the recording is over, you can terminate the session by using ***exit*** command.

$exit
Script done, file is typescript
$

To view the file *typescript*, one can use ***cat*** command.

Note that, the usage of *script* command overwrites any existing file with name *typescript.* If you want to append the new content to existing file, then use ***–a*** as below –

$script –a

Now, the previous *typescript* will be appended with the activities of this session.

If you want to create your own file instead of *typescript* file, then give the required filename as –

$script mylogfile

Now, the activities of this session will be stored in the file *mylogfile.*

NOTE that, some activities like the commands used in the full-screen mode like ***vi*** editor will not be recorded properly when we record session using *script* command.

spell

The ***spell*** command is used to check the spelling in a text file. When the name of a file is given an argument to this command, it lists out all the mistakes (words without proper meaning as per the understanding of UNIX). To understand this command, let us first create a file as below –

$cat >test.txt
hello hw are yu?
Im doin fine

Now, apply ***spell*** command on the file *test.txt* as shown –

$ spell test.txt
doin
hw
Im
yu

One can observe that the words with spelling mistakes have been displayed in the order.

Now, if you want to correct these words, ***ispell*** command can be used. ***ispell*** is actually an interactive editor, which displays various suggestions for a mistaken word. Then, user has to

choose one of the possible suggestions listed and the mistaken word will be replaced by the corrected word. For example,

$ispell test.txt

uname

The command ***uname*** is a short-form for UNIX name, which displays the details like name and version of the machine and OS currently running. It can display various details based on the option given to it as an argument. Consider following situations:

- $uname

Linux

The command without any options displays the name of underlying OS.

- $uname –a

Linux server4 2.6.18-128.el5xen #1 SMP Wed Dec 17 12:01:40 EST 2008 x86_64 x86_x

This has displayed details like kernel name, node name, kernel release, kernel version etc.

- $uname –n

server4

When your system is connected to network, it prints the name of the machine in network. This name is required while copying the files from remote machine using ***ftp*** command.

bc

UNIX provides two types of calculators – a graphical (GUI) calculator (similar to the one available in windows OS) and a character based ***bc*** command. A visual calculator can be available using ***xcalc*** command and it is available only on X Window system, but not on command-line based terminals.

The calculator available through ***bc*** command is a very powerful, but sadly a most neglected tool in UNIX. When ***bc*** command is invoked without any argument, it does nothing but waits for the input from the keyboard. Once the job is done, *ctrl+d* has to be pressed to release the command and to get a prompt.

The usage of ***bc*** command is illustrated here with examples.

- **Basic operations:**

$bc
3+5
8
5*6
30
6-10
-4
[ctrl+d]

- **To perform more than one operation in a single line:**

$bc
2^4; 3+6 //using semicolon as a separator
16
9
[ctrl+d]

- **Setting scale for required precision during division operation:**

By default, ***bc*** performs truncated division (or integer division). For example,

$bc
9/5
1

Here, the output 1, instead of 1.8. To avoid such truncation, one can set the precision after the decimal point. For example,

$bc
scale=2
9/5
1.80

22/7
3.14

 Converting numbers from one base to the other:

One can change the base of a number by setting ***ibase*** (input base) or ***obase*** (output base). For example –

```
$bc
ibase=2 //setting input base as 2
1100
12 //decimal equivalent of 1100
11001110
206 //decimal equivalent of 11001110
```

The reverse is possible through ***obase*** as shown below –

```
$bc
obase=2 //setting output base as 2
14
1110 //binary equivalent of 14
308
100110100 //binary equivalent of 308
obase=16 //setting output base as 16 (hexa)
14
E //hexadecimal equivalent of 14
```

 Storing variables:

One can store values in variables and then use them. But, ***bc*** supports only single lowercase letters (a-z) and hence, one can use only 26 variables at a time. For example –

```
$bc
a=5; b=3; c=2
p=a+b*c
p //use variable name to display the result
11
```

Note that, ***bc*** is a pseudo-programming language that supports arrays, functions, conditional structures (if) and looping structures (for and while). It also supports library of some scientific functionalities. It can handle very large numbers. If the result of some calculation is 900 digits, the ***bc*** command shows every digit of it!!

The reverse is possible through ***obase*** as shown below –

```
$bc
obase=2 //setting output base as 2
14
1110 //binary equivalent of 14
308
100110100 //binary equivalent of 308
obase=16 //setting output base as 16 (hexa)
14
E //hexadecimal equivalent of 14
```

,

$who

Normally a UNIX system is used by multiple users at a time. One user may needs to know the list of other users who are using the system currently. The *who* command is used for this purpose. This command displays name of the users (login ID used to log in), name of the terminal and date and time of login. For example –

```
$who
root :0 Sept 04 10:12
chetana tty01 Sept 04 11:11
raghu tty02 Sept 04 12:35
ram tty03 Sept 04 14:08
```

Here the first column shows the user-ids of four users who are currently logged in. In the second column, *tty01* etc. are name of the terminals and the last column shows date and time of their

respective login. This indicates that currently (while giving *who* command), four users have logged in. The term *tty* indicates *teletype.* The machine identifies a person with his/her username. So, user will be the owner of file he has created. When a file created by one user, say *chetana* is sent to another user, the machine will inform the recipient that a mail has arrived from *chetana.*
Some of the systems display as below when *who* command is used –
$who
chetana pts/1 Sept 04 11:11
Here, pts/1 is the name of the terminal. The term *pts* stands for *pseudo-terminal slave.*
Most of the UNIX/Linux systems have software implementation as an interface to interact with real terminal. It is called as *pseudo-terminal* or *pseudo-teletype* represented by *pty.*
The *pts* is a slave part of *pty.*
The header option *–H* can be used along with *–u* option to get more information on the ***who*** command.
$who –Hu
NAME LINE TIME IDLE PID COMMENT
root :0 2007-01-12 04:49 ? 5595
chetana pts/1 2007-01-13 05:39 . 24081 (172.16.4.205)
Here, first three columns are same as before. The fourth column *IDLE* indicates from how long the user is idle. The dot (.) indicates the respective user was active in the last one minute. The question mark (?) indicates that the user is idle from quite a long time, which is unknown. The fifth column PID (process identifier) will be discussed in later chapters. The comment line indicates some special comment, if any. In the above example, for the user *chetana*, it is showing the IP address of the machine.

2.a State and explain the types of shell variables in UNIX.

Shell variables are of 2 types
Local variables
Environment Variables
Local Variables:
They are more restricted in scope
Ex: DOWNLOAD_DIR=/home/kumar/download
echo $DOWNLOAD_DIR
Environment Variables:
They are available in the users total environment i.e., the sub shells tha run shell scripts and mail commands and editors.
The common environment variables are

Variable	Significance
HOME	Home directory-the directory a user is placed on logging in
PATH	List of directories searched by shell to locate a command
LOGNAME	Login name of user
USER	Login name of user
MAIL	Absolute pathname of users mailbox office
MAILCHECK	Mail checking interval for incoming mail
TERM	Type of terminal
PWD	Absolute pathname of current directory
CDPATH	List of directories searched by cd when used with a non absolute path name.
PS1	Primary prompt string
PS2	Secondary prompt string
SHELL	Users login shell and one invoked by programs having shell escapes

2.b Write a note on here document.

Sometimes, the shell uses the << symbols to read data from the same file containing the

script. This is referred to as here document, indicating that the data is here only, not in a separate file. Any command using standard input can also take input from a here document.
Consider an interactive script, that is, a shell script which reads some input from the keyboard.

hereDoc.sh

```
#!/bin/sh
#Illustration of here document

echo "Enter your name:"
read fname
echo "Enter your age:"
read age

echo "Your name is $fname, Your age is $age"
```

When we run this script in a normal way, it would look something like this –

$ sh hereDoc.sh
Enter your name:
Ramu
Enter your age:
21
Your name is Ramu, Your age is 21
Now, let us see, how to use ***here document*** for this script. Run the above script as shown below –
$ sh hereDoc.sh <<END
>Ramu #shell waits for your input from this line
>21
>END # till this line
Enter your name:
Enter your age:
Your name is Ramu, Your age is 21
Observe the above lines. While running the script, we have used a term <<END. Here, the symbol << indicates that the file *hereDoc.sh* will be reading an input from the ***here document*** but not from the keyboard. The word *END* used is just an example for delimiter, and one can use any word (not UNIX command). After the first line, user can keep giving the inputs. Once the input is done, the delimiter has to be provided. Immediately after seeing the delimiter word for the second time, the *hereDoc.sh* file starts executing and the *read* commands inside the script will not wait for the user input from the keyboard, instead, it will be taken from the ***here document*** created already.

3. With a neat diagram, explain the architecture of Unix operating system.

UNIX OS distributes its major jobs into two agencies viz. kernel and shell.

Kernel: The kernel is also known as operating system. It interacts with the hardware of the machine. The kernel is the core of OS and it is a collection of routines/functions written in C. Kernel is loaded into memory when the system is booted and communicates with the hardware. The application programs access the kernel through a set of functions called as system calls. The kernel manages various OS tasks like memory management, process scheduling, deciding job priorities etc. Even if none of the user programs are running, kernel will be working in a background.

Shell: The shell interacts with the user. It acts as a command interpreter to translate user's command into action. It is actually an interface between user and the kernel. Even though there will be only one kernel running, multiple shells will be active – one for each user. When a command is given as input through the keyboard, the shell examines the command and simplifies it and then communicates with the kernel to see that the command is executed. The shell is represented by sh (Bourne Shell), csh (C Shell), ksh (Korn shell), bash (Bash shell).

The relationship between kernel and shell is shown in Figure

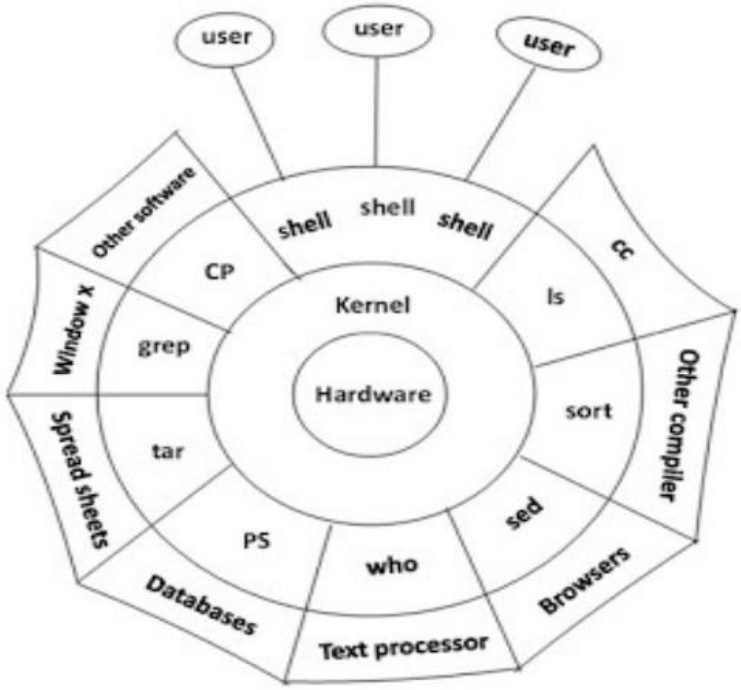


Figure 1.1 The Kernel – Shell Relationship

The File and Process

The file and the process are two simple entities that support UNIX.

File: A file is an array of bytes and it can contain any data. Various files are related to each other by a hierarchical structure. Even a user (user name) is placed in this file system. UNIX considers directories and the devices also as the members of file system. In UNIX, the major file type is text and the behavior of UNIX is controlled mainly by text files. UNIX provides various text manipulation tools through which the files can be edited without using an editor.

Process: A processing a program under execution. Processes are also belonging to separate hierarchical structure. A process can be created and destroyed. UNIX provides tools to the user to control the processes, move them between foreground and background and to kill them.

The System Calls:

System calls are used to communicate with the kernel. There are more than thousand commands in UNIX, but they all use few set of function called as system calls for communication with kernel. All UNIX flavors (like Linux, Ubuntu etc) all use the same system calls. For example, write is a system call in UNIX. C programmer in UNIX environment can directly use this system call to write data into a file. Whereas, the C Programmer in Windows environment may need to use library function like fprintf() to write into a file. A system call open in UNIX can be used to open a file or a device. Here, the purpose is different, but the system call will be same. Such feature of UNIX allows it to have many commands for user purpose, but only few system calls internally for the actual work to be carried out in association with the kernel.

4.a. Briefly explain for loop in shell script with examples.

It is very important to note that (especially those who know higher programming languages) the ***for*** loop in shell script is NOT same as that in other languages. One can neither increment/decrement the values, nor specify the condition to be met. Instead, it just iterates over the elements in a list. A set of commands are executed until the list gets exhausted.

The syntax is –

for variable in list
do
execute commands
done

Consider an example –

Write a shell script to find sum of numbers provided through command line.

sum.sh
Output:
While running this script, give command line arguments similar to –
$ sh sum.sh 10 20 30
Sum=60

4.b Explain different forms of if statement with an example.

One of the important requirements in programming is conditional structures. In shell programming, the conditional construct ***if*** can be used in the following ways –

if **statement:**

```
if command is successful
then
execute commands
fi
```

if else statement:

```
if command is successful
then
execute commands
else
execute commands
fi
```

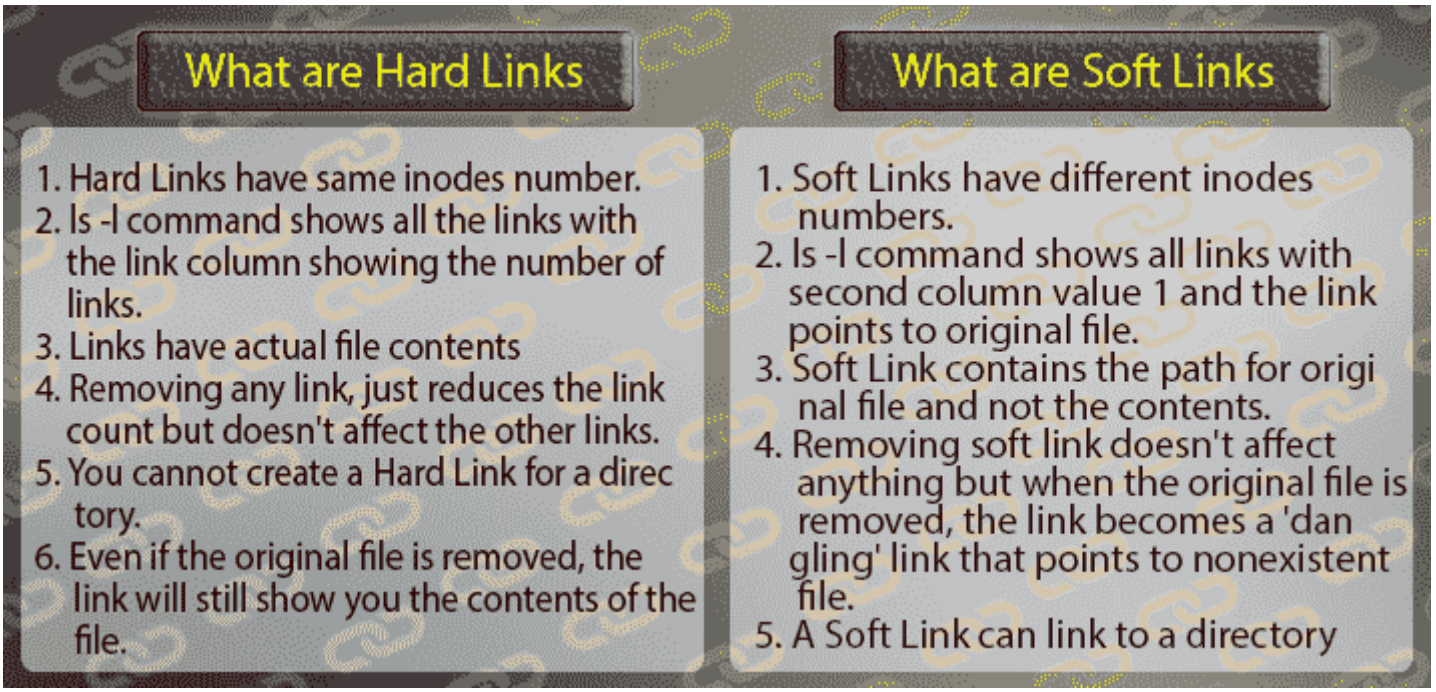
elif statement:

```
if command is successful
then
execute commands
elif command is successful
then
execute commands
elif command is successful
then
…….
else
……..
fi
```

Ex:

```
#!/bin/sh
#Illustration of if statement
x=5
y=10
if test $x -lt $y
then
echo "$x is less than $y"
else
echo "$y is less than $x"
fi
if [ $x -ne $y ]; then
echo "$x and $y are not equal"
fi
```

5.a Differentiate between hard link and soft link.



5.b With suitable example bring out the difference between absolute and relative path names.

ABSOLUTE PATH

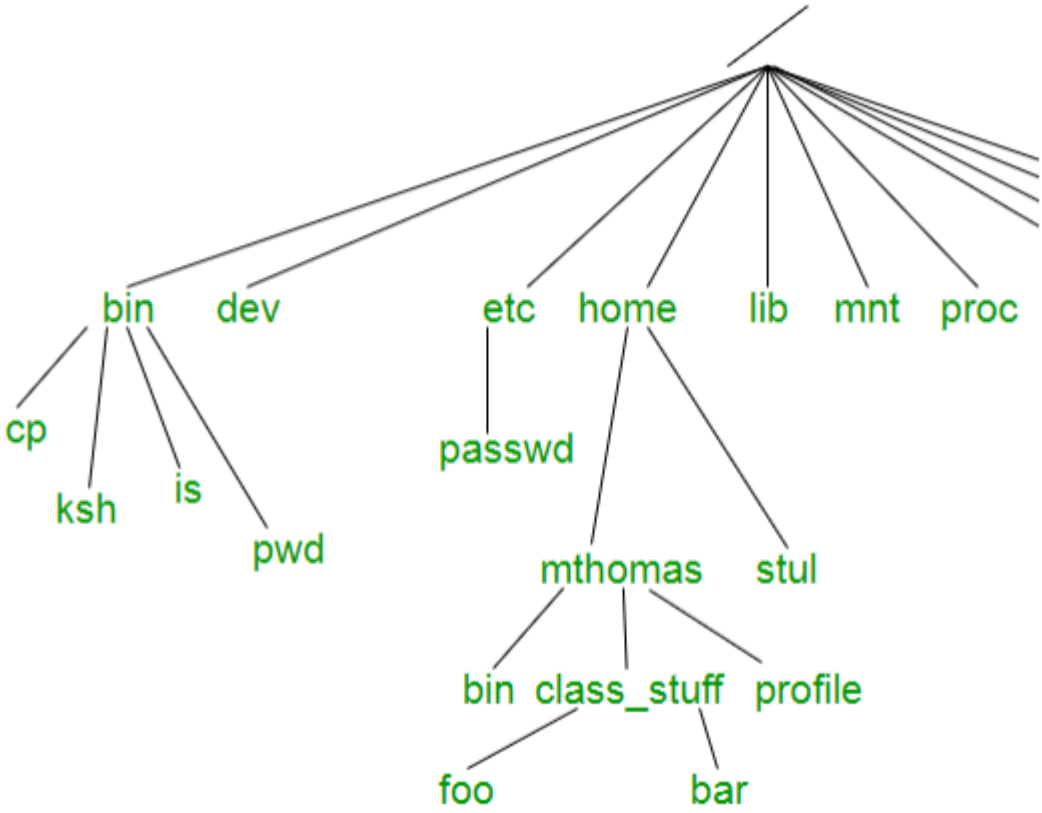
VERSUS

RELATIVE PATH

Absolute Path	Relative Path
It points to a specific location in the file system, irrespective of the current working directory.	It points to the location of a directory using current directory as reference.
It is also referred to as full path or file path.	It is also referred to as non-absolut path.
It refers to the location of a file or directory (filesystem) relative to the root directory in Linux.	It refers to the location of a file or directory (filesystem) relative to th current directory.
Absolute URLs are used to link to other websites that are not located on the same domain.	Relative URLs are used to link to other websites that are located on the same domain.
For example: If your pictures are in C:\Sample\Pictures and index in C:\Sample\Index, then the absolute path for pictures is C:\Sample\ Pictures.	For example: If your pictures are i C:\Sample\Pictures and index in C Sample\Index, the relative path is "..\Pictures".

6.a Briefly explain the unix file system.

Unix file system is a logical method of organizing and storing large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. Unix file system has several important features. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system. Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called "root" which is represented by a "/". All other files are "descendants" of root.



Directories or Files and their description –

• / : The slash / character alone denotes the root of the filesystem tree.

• /bin : Stands for "binaries" and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.

• /boot : Contains all the files that are required for successful booting process.

• /dev : Stands for "devices". Contains file representations of peripheral devices and pseudo-devices.

• /etc : Contains system-wide configuration files and system databases. Originally also contained "dangerous maintenance utilities" such as init, but these have typically been moved to /sbin or elsewhere.

• /home : Contains the home directories for the users.

• /lib : Contains system libraries, and some critical files such as kernel modules or device drivers.

• /media : Default mount point for removable devices, such as USB sticks, media players, etc.

• /mnt : Stands for "mount". Contains filesystem mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) filesystems, CD-ROM/DVD drives, and so on.

• /proc : procfs virtual filesystem showing information about processes as files.

• /root : The home directory for the superuser "root" – that is, the system administrator. This account's home directory is usually on the initial file system, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.

• /tmp : A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.
• /usr : Originally the directory holding user home directories,its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of /usr, such as the default as in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person).
• /usr/bin : This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.
• /usr/include : Stores the development headers used throughout the system. Header files are mostly used by the #include directive in C/C++ programming language.
• /usr/lib : Stores the required libraries and data files for programs stored within /usr or elsewhere.
• /var : A short for "variable." A place for files that may change often – especially in size, for example e-mail sent to users on the system, or process-ID lock files.
• /var/log : Contains system log files.
• /var/mail : The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.
• /var/spool : Spool directory. Contains print jobs, mail spools and other queued tasks.
• /var/tmp : A place for temporary files which should be preserved between system reboots.

6.b What is a file? Explain the categories of files found in UNIX Operating System.

The file is a container for storing information. Unlike old DOS files, a UNIX file does notcontain eof (end-of-file) character. It contains only the information stored by the user. All the attributes of a file are kept in a separate area of the hard disk, which can be accessible by only the kernel. UNIX treats directories and devices also as files. Even the physical devices like hard disk, memory, CD-ROM, printer, modem etc. are treated as files.
In UNIX, files are divided into three categories –
- Ordinary file
- Directory file
- Device file

These three types of files are discussed in detail in the following sections –

2.2.1 Ordinary (Regular) File

It is a most common type of file that contains only data as a stream of characters. An ordinary file can be one among these –
- **Text file:** contains only printable characters. Source codes of programming languages like C, Java, C++, Perl, Shell script etc. are all text files. A text file contains lines of characters where every line is terminated with a *newline* character – known as *linefeed (LF)*. Whenever you press [Enter] key while inserting text into a file, the LF character is appended. One cannot see this character, but it can be made visible using the command ***od***.
- **Binary file:** contains both printable and non-printable characters covering entire ASCII (0 – 255) set. The object codes, executable files etc. created by compiling C language are binary files. Most of the UNIX commands are binary files. Image/audio/video files are binary files. Trying to display the contents of such files using simple *cat* command would produce unreadable output.

2.2.2 Directory File

A directory contains no data, but it keeps some information about the files and subdirectories that it contains. The UNIX file system is organized with a number of directories and subdirectories. A user also can create them, as and when required. Usually, a group of related files are kept in a single directory. Sometimes, files with same name are kept in different directories.
A directory file contains an entry for every file and subdirectory it has. Each such entry has two components viz. –
- The filename
- A unique identification number for the file or directory (called as the *inode number*)

Thus, a directory actually do not *contain* the file itself, rather, it contains only the file name and a number.
One cannot write into a directory file. But, the actions like creating a file, removing a file etc. makes kernel to update the corresponding directory by creating/removing filename and inode number associated with that file.

2.2.3 Device File

The activities like printing files, installing softwares from CD-ROM, taking backup of files into a tape/drive etc. are performed by reading or writing the file representing the device. For example, when you are printing a file in a printer, you are writing a file associated with printer.
Device filenames are generally found inside a single directory structure, /dev. A device file is not a stream of characters. In fact, it does not contain anything. The operation of a device is completely managed by the attributes of its associated file. The kernel identifies a device from its attributes and then uses them to operate the device.

7. Write a shell script to find the given input character is an uppercase, small case, digit or special symbol using case conditional statement.

```
printf 'Please enter a character: '
IFS= read -r c
case $c in
  ([[:lower:]]) echo lowercase letter;;
  ([[:upper:]]) echo uppercase letter;;
  ([[:alpha:]]) echo neither lower nor uppercase letter;;
  ([[:digit:]]) echo decimal digit;;
  (?) echo any other single character;;
  (*) echo anything else;;
esac
```

8.a. What are attributes? How to list file attributes and directory attributes? Discuss with examples.

Every file is associated with a table that contains various attributes of a it, except its name and contents. This table is called as ***inode*** (index node) and accessed by the ***inode number***. The ***inode*** contains following attributes of a file –

- File type (ordinary, directory, device etc)
- File permissions
- Number of links
- The UID of the owner
- The GID of the group owner
- File size in bytes
- Date and time of last modification
- Date and time of last access
- Date and time of last change of the inode
- An array of pointer that keep track of all disk blocks used by the file

ls –l option: Listing File Attributes

The –l option of ***ls*** command is used for listing the various attributes like permissions, size, ownership etc. of a file. The output of ***ls –l*** is referred to as the ***listing***. The –l option can be combined with other options for displaying other attributes, or ordering the list in a different sequence. The command ***ls*** use inode of a file to fetch its attributes. Consider the following example of ***ls –l*** which displays seven attributes of all files in the current directory.

```
$ ls -l
total 144
-rw-rw-r-- 1 john john 280 Jan 30 09:56 caseEx.sh
-rw-rw-r-- 1 john john 104 Feb 3 06:40 cmdArg.sh
-rw-rw-r-- 1 john john 199 Jan 29 10:58 ifEx.sh
-rw-rw-r-- 1 john john 217 Jan 19 09:25 logfile
drwxrwxr-x 2 john john 4096 Feb 6 05:48 myDir
-rwxrwxr-x 1 john john 29 Jan 22 10:04 myFirstShell
-rw-rw-r-- 1 john john 43 Jan 22 10:44 second.sh
```

The output of ***ls –l*** starts with total 144, indicates that a total of 144 blocks are occupied by these files on disk. The 7 types of attributes/fields displayed by the command are discussed below –

- **File Type and Permissions:** The first column shows the type and permissions associated with each file. If the first character in this column is – (hypen), then it is an ordinary file. On the other hand, if the first character is d, then it is a directory. Then, there is a series of r, w, x and – (hyphen) indicating file permissions read, write and execute. The hyphen indicates absence of particular permission.
- **Links:** The second column indicates the number of links associated with the file. It is a number of filenames maintained by the system of that file. Usually for ordinary files, it will be just 1. But for directories, it will be number of files contained within that directory (including current directory).
- **Ownership:** The creator of the file would be its owner. In the third column, it shows john as the owner. The owner has full authority to modify the contents and permissions of a file. Similarly, the owner of a directory can create modify or remove files in that directory.
- **Group Ownership:** While opening a user account, a system administrator assigns the user to some group. The fourth column represents the group owner of that file.
- **File Size:** Size of the file in bytes is shown as fifth column. The size is only a character count of the file, but not the amount of space it occupies in the disk.
- **Last Modification Time:** The 6th, 7th and 8th columns indicate the last modification time of the file. A file is said to be modified only if its contents have changed. If you change only the permission or ownership of the file, its last modification time field will not get affected.
- **Filename:** The last field is the name of the file, usually in ASCII collating sequence.

ls –d option: Listing Directory Attributes

If we want to list the attributes of only the directory, but not its contents, we can use –d option as below –

$ ls –ld myDir

drwxrwxr-x 2 john john 4096 Feb 6 05:48 myDir

8.b. Explain how one can change the permission of a file in detail.

A file or directory is created with a default set of permissions. Generally, in the default setting, write permission is not given to group and others. That is, only the user (owner) the file can write a file. But, read permission will be given to all. The ***chmod*** (change mode) command is used for assigning/removing different permissions to/from *category* (user, group, others). This command can be run only by the user (owner) and the super-user (admin). The ***chmod*** command can be used in two ways –

- In a relative manner by specifying the changes to the current permissions
- In an absolute manner by specifying the final permissions

Consider the permissions of an existing file *test* as below –

$ls –l test

-rw-r--r-- 1 john richard 853 Sep 5 23:38 test

It is observed here that, by default the execute permission is not there even for the user (owner). Keeping this status of the file *test* as a base, let us discuss different ways of using ***chmod*** command.

Relative Permissions

When changing permissions in a relative manner, ***chmod*** changes only the permissions specified in the command line and leaves the other permissions unchanged. The syntax is–

chmod *category operation permission filenames*

The argument for ***chmod*** is an expression consisting of some letters and symbols describing user category and type of permission being assigned/removed. The expression contains three components:

- User category (user: **u**, group: **g**, others: **o**, All: **a**)
- The operation to be performed (assign: **+**, remove: **–**, assign absolute permission: **=**)
- The type of permission (read: **r**, write: **w**, execute: **x**)

Now, consider the example of the file *test* taken before, and assign execute permission to it as below –

$ chmod u+x test #assign(+) x(execute) to u(user)

$ ls –l test

-rwxr--r-- 1 john richard 853 Sep 5 23:38 test
Now, the user *john* got permission to execute the file *test.* If you want to assign execute permission on file *test* to group and others also, then use the command as –
$ chmod ugo+x test #assign(+) x to u(user, group, others)
$ ls –l test
-rwxr-xr-x 1 john richard 853 Sep 5 23:38 test
The string ugo can be replaced by *a* indicating *all* as shown below –
$ chmod a+x test #assign(+) x to a(all)
$ ls –l test
-rwxr-xr-x 1 john richard 853 Sep 5 23:38 test
When you are willing to assign a particular permission to *all*, then even the character *a* can be omitted as below –
$ chmod +x test #assign(+) x to all
When same set of permissions has to be assigned to more than one file, then we can give multiple files separated with space as –
$ chmod u+x test test1 test2
To remove a permission, the – (hyphen or minus) operator is used. For example, to remove read permission from group and other, we can do as below –
$ ls –l test #check current status
-rwxr-xr-x 1 john richard 853 Sep 5 23:38 test
$ chmod go-r test #remove r permission from group & others
$ ls –l test #verify
-rwx—x--x 1 john richard 853 Sep 5 23:38 test
Multiple expressions separated by comma can be given to ***chmod*** command. For example, to remove the execute permission from all and then to assign read permission to group and others, a single statement can be used as –
$ chmod a-x, go+r test
$ ls –l test
-rw-r--r-- 1 john richard 853 Sep 5 23:38 test
More than one permission can also be set as below –
$chmod o+wx test
$ ls –l test
-rw-r--rwx 1 john richard 853 Sep 5 23:38 test
Here, write and execute permissions are set to *others*.

Absolute Permissions

Irrespective of existing permissions for a file, we may need to assign a new set of permissions. That is, we wish to set all nine permission bits explicitly. This is known as *absolute permissions*. For this purpose, ***chmod*** uses a string of three octal numbers. Various permissions are given a specific digit as below –

- Read permission – 4
- Write permission – 2
- Execute permission – 1

Every possible combination of three different permissions is shown in binary representation in Table 2.1.

Table 2.1 Digits used for absolute pathnames

Binary Octal Permission Significance

000 0 - - - No permission
001 1 - - x Execute only
010 2 - w - Write only
011 3 - wx Write and execute
100 4 r- - Read only
101 5 r – x Read and execute
110 6 rw- Read and write
111 7 rwx Read, write and execute

Now, let us see some examples of using the absolute permissions with the help of octal digits.

Ex 1. Assigning read and write(4+2=6) permissions to all –
$ chmod 666 test
$ ls –l test

-rw-rw-rw- 1 john richard 853 Sep 5 23:38 test
Ex 2. To remove the write permission from group and others:
$ chmod 644 test
$ ls –l test
-rw-r--r-- 1 john richard 853 Sep 5 23:38 test
Note that, there is nothing like removing some permission. It is just reassignment of new set of permissions to all.
Ex 3. To assign all permissions to owner, read and write permissions to group and only execute permission to others –
$ chmod 761 test
$ ls –l test
-rwxrw---x 1 john richard 853 Sep 5 23:38 test

Using chmod Recursively (–R)

The *chmod* command can be used to apply required permissions on all files (and files within subdirectory) in a given directory. It is done using –R option as below –
$chmod –R a+x ShellPgms
This makes all files and subdirectories found in the tree-walk (starting from ShellPgms directory, includes all files in subdirectories) executable by all users. One can provide multiple directory and filenames for this purpose. If *chmod* has to be applied on home directory tree, one can use any one of the following –
$chmod –R 755 . #works on hidden files also
$chmod –R a+x * #leaves out hidden files

9. Discuss the types of tests performed by the test command.

Evaluate a boolean expression setting the Exit Code to indicate a *true* or *false* result. This is used to express the logic of the Control Constructs used for shell script programming.

Note from the synopsis that there are two ways to invoke **test** – either with the command or the alternate form using square brackets. The square brackets have the advantage of giving a more familiar look, but one must be careful to **leave spaces between the brackets and the boolean expression**.

SYNOPSIS

test *expression*

[*expression*]

String Comparisons

-n string	True if length of string is non-zero
-z string	True if length of string is zero
string1 == string2	True if the strings are equal
string1 != string2	True if the strings are not equal

Much of shell script programming often relates to working with files and directories, so the following boolean expressions are frequently used.

File Oriented Expressions

-a *file*	True if the file exists
-b *file*	True if the file exists and is a block-oriented special file
-c *file*	True if the file exists and is a character-oriented special file
-d *file*	True if the file exists and is a directory

-e *file*	True if the file exists
-g *file*	True if the file exists and its "set group ID" bit is set
-p *file*	True if the file exists and is a named pipe
-r *file*	True if the file exists and is readable
-s *file*	True if the file exists and has a size greater than zero
-t *fd*	True if the file descriptor is open refers to the terminal
-u *file*	True if the file exists and its "set user ID" bit is set
-w *file*	True if the file exists and is writable
-x *file*	True if the file exists and is executable
-O *file*	True if the file exists and is owned the effective user ID of the user.
-G *file*	True if the file exists and is owned the effective group ID of the user.
-L *file*	True if the file exists and is a symbolic link
-N *file*	True if the file exists and has been modified since it was last read
-S *file*	True if the file exists and is a named socket
file1 -nt *file2*	True if *file1* is newer than *file2*
file1 -ot *file2*	True if *file1* is older than *file2*
file1 -ef *file2*	True if *file1* and *file2* have the same device and inode numbers

Numeric Comparisons:

Operator	Meaning
-eq	equal to
-ne	not equal to
-gt	greater than
-ge	greater than or equal to
-lt	less than
-le	less than or equal to

10. Describe the important features of the UNIX operating system

A Multiuser System: UNIX is basically a multiprogramming system. Here, either Multiple users can run separate jobs or Singe user can run multiple jobs. In UNIX, many processes are running simultaneously. And, the resources like CPU, memory and hard disk etc are shared between all users. Hence, UNIX is a multiuser system as well. The Unix system breaks up one time unit into several segments and each user is allotted one segment. At any point of time, the machine will be doing the job of one user. When the allotted time expires, the job is temporarily suspended and next user's job is taken up. This process continues till all processes gets one segment each and once again the first user's job is taken up. The kernel does this task several times in one second such a way that the users will never come to know about it and users cannot make out the delay in between.

A Multitasking System: Unix is a multitasking system, wherein a single user can run multiple jobs concurrently. A user may edit a file, print a document on a printer and open a browser etc – all at a time. In multitasking environment, a user can see one job running in the foreground and all other jobs run in the background. The jobs can be switched between background and foreground; they can be suspended or terminated.

The Building-Block Approach: Unix is a collection of few hundred commands, each of which is designed to perform one task. More than one command can be connected via the | (pipe) symbol to perform multiple tasks. The commands which can be connected are called as filters because, they filter or manipulate data in different ways. Many Unix tools are designed such a way that the output of one tool can be used as input to another tool. For this reason, UNIX commands do not generate lengthy or messy outputs. If a program is interactive, then user's response to it may be different. In such situations, the output of one command cannot be made as input to another command. Hence, UNIX programs are not interactive.

The UNIX Toolkit: Unix contains diverse set of applications like text manipulation utilities, compilers and interpreters, networked applications, system administration tools etc. The Unix kernel does many tasks with the help of these applications. Such set of tools are constantly

varying with every release of UNIX. In every release, new tools are being added and old tools are either removed or modified. Most of these tools are open-source utilities and the user can download them and configure to run on one's machine.

Pattern Matching: Unix has very sophisticated pattern matching features. The character like * (known as a metacharacter) helps in searching many files starting with a particular name. Various characters from a metacharacter set of Unix will help the user in writing regular expressions that will help in pattern matching.

Programming Facility: The Unix shell is a programming language as well. It provides the user to write his/her own programs using control structures, loops, variables etc. Such programs are called as shell scripts. Shell scripts can invoke Unix commands and they can control various functionalities of Unix OS.

Documentation: Unix provides a large set of documents to understand the working of every command and feature of it. The man command can be used on an editor to get the manual about any Unix command. Moreover, there are plenty of documents, newsgroups, forums and FAQ (Frequently Asked Questions) files available on internet, where one can get any information about Unix.